

Symbolic Execution & Program Repair

Jinbo Du, 2015/12/23

Motivation

- Symbolic execution
 - Generate test cases
- Program repair
 - Test Suite Based Repair: Fault localization & Patch validation

Motivation

- Symbolic execution
 - Generate test cases
- Program repair
 - Test Suite Based Repair: Fault localization & Patch validation

=>

- Generate test cases to improve the “effectiveness” for program repair

Related Work: Program Repair

- Defects4j (Ready)
 - “Defects4J: A database of existing faults to enable controlled testing studies for Java programs” by René Just, Darioush Jalali, and Michael D. Ernst. ISSTA’14
 - Defects4J by Just et al. is a bug database that consists of 357 real-world bugs from five widely-used open-source Java projects
 - <https://github.com/rjust/defects4j/>

Related Work: Program Repair, cont.

- Defects4j-Repair (runs on Grid'5000)
 - “Automatic Repair of Real Bugs: An Experience Report on the Defects4J Dataset” by Thomas Durieux, Matias Martinez, etc. Arxiv, 2015.
 - “all previous evaluations of automatic repair techniques were made on a bug dataset that was specifically built for the evaluation of those techniques”
 - 3 tools implementation & evaluation
 - jGenProg, Kali, Nopol
 - fixed together 41 of 224 (18%), with 59 patches
 - 8/42 out of 59 patches are correct: test suites are weak!

Related Work: Program Repair, cont.

- GenProg & Kali Implementation: Astor Framework

- Matias Martinez, Martin Monperrus. "ASTOR: Evolutionary Automatic Software Repair for Java". Technical Report hal-01075976, Inria; 2014. Matias Martinez, Martin Monperrus. "ASTOR: Evolutionary Automatic Software Repair for Java". Technical Report hal-01075976, Inria; 2014.
- <https://github.com/SpoonLabs/astor>

- Nopol Implementation

- F. DeMarco, J. Xuan, D. Le Berre, and M. Monperrus. Automatic repair of buggy if conditions and missing preconditions with SMT. In Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, pages 30–39. ACM, 2014.
- <https://github.com/SpoonLabs/nopol>

Related Work: Symbolic Execution

- Java PathFinder
- Symbolic JPF
 - “Combining Unit-level Symbolic Execution and System-level Concrete Execution for Testing NASA Software” Corina Păsăreanu, Peter Mehlitz, David Bushnell, Karen Gundy-Burlet, Michael Lowry (NASA Ames) Suzette Person (University of Nebraska, Lincoln) Mark Pape (NASA JSC), ISSTA’08
- Concolic walk
 - Peter Dinges and Gul Agha. “Solving Complex Path Conditions through Heuristic Search on Induced Polytopes”. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne Storey, editors, FSE 2014
 - <https://github.com/osl/concolic-walk>

Defects4j - Example 1

```
@Test
public void testIsEndedBy() {
    assertFalse(intRange.isEndedBy(null));

    assertFalse(intRange.isEndedBy(5));
    assertFalse(intRange.isEndedBy(10));
    assertFalse(intRange.isEndedBy(15));
    assertTrue(intRange.isEndedBy(20));
    assertFalse(intRange.isEndedBy(25));
}

@Test
public void testIsBefore() {
    assertFalse(intRange.isBefore(null));

    assertFalse(intRange.isBefore(5));
    assertFalse(intRange.isBefore(10));
    assertFalse(intRange.isBefore(15));
    assertFalse(intRange.isBefore(20));
    assertTrue(intRange.isBefore(25));
}
```


Defects4j - Example 2

```
@Test
public void testComparableConstructors() {
    final Comparable c =
        new Comparable() {
            @Override
            public int compareTo(final Object other) {
                return 1;
            }
        };
    final Range r1 = Range.is(c);
    final Range r2 = Range.between(c, c);
    assertEquals(true, r1.isNaturalOrdering());
    assertEquals(true, r2.isNaturalOrdering());
}
```

Defects4j - Example 3

```
protected void testCreateBigIntegerFailure(final String str) {
    try {
        final BigInteger value = NumberUtils.createBigInteger(str);
        fail("createBigInteger(\"" + str + "\") should have failed: " + value);
    } catch (final NumberFormatException ex) {
        // empty
    }
}

@Test
public void testCreateBigDecimal() {
    assertEquals("createBigDecimal(String) failed", new BigDecimal("1234.5"), Nu
    assertEquals("createBigDecimal(null) failed", null, NumberUtils.createBigDec
    this.testCreateBigDecimalFailure("");
    this.testCreateBigDecimalFailure(" ");
    this.testCreateBigDecimalFailure("\b\t\n\f\r");
    // Funky whitespaces
    this.testCreateBigDecimalFailure("\u00A0\uFEFF\u000B\u000C\u001C\u001D\u001E
    this.testCreateBigDecimalFailure("-"); // sign alone not valid
    this.testCreateBigDecimalFailure("--"); // comment in NumberUtils suggests s
    this.testCreateBigDecimalFailure("--0");
    this.testCreateBigDecimalFailure("+"); // sign alone not valid
    this.testCreateBigDecimalFailure("++"); // in case this was also allowed by
    this.testCreateBigDecimalFailure("++0");
}
```

Approach: Generate tests from existing

- Generate test inputs
 - Make existing test inputs symbolic
 - run JPF to generate new inputs
- Generate test oracles - Challenge
 - on buggy programs
 - syntheses existing test oracles

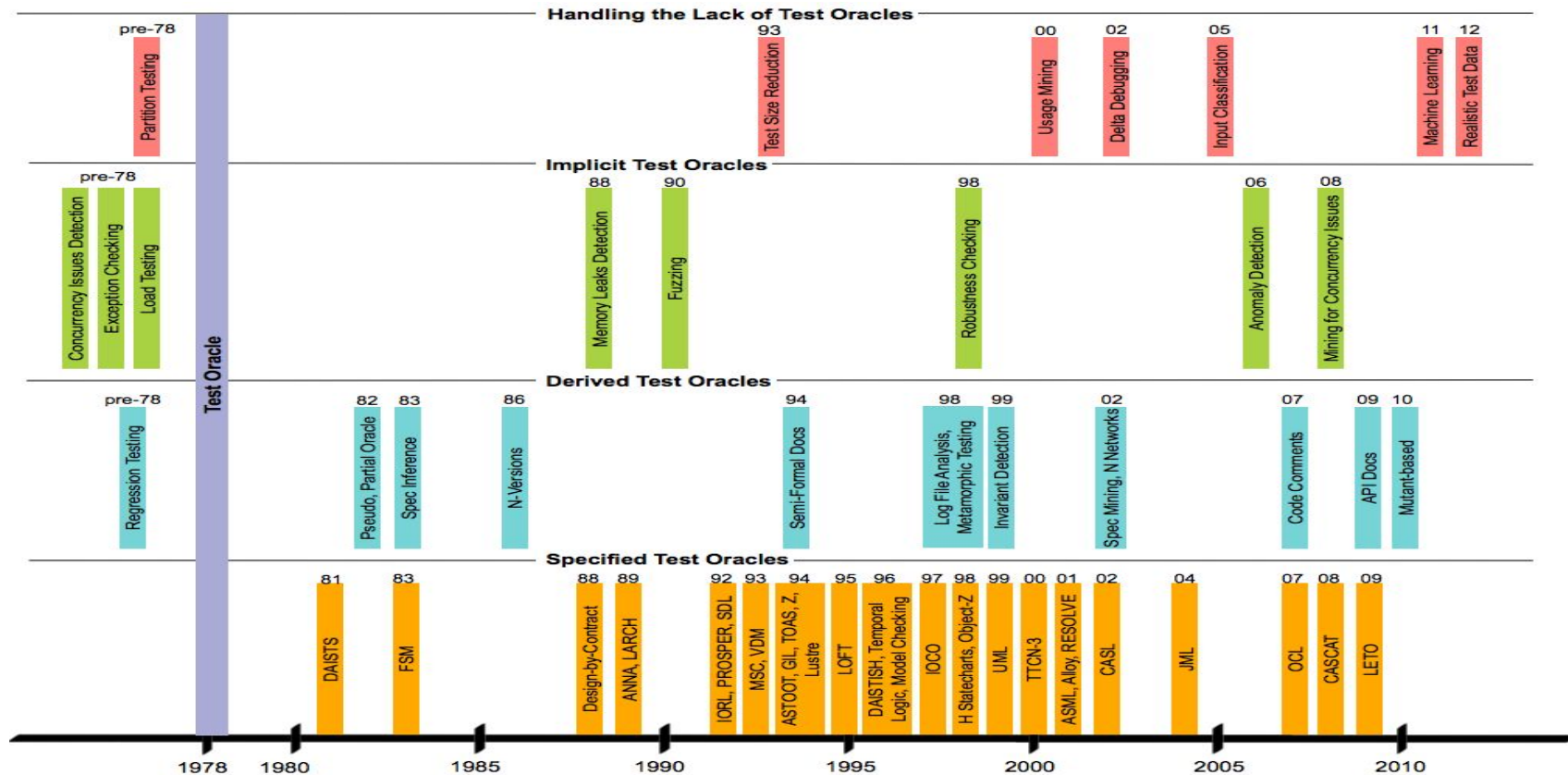
Challenge: “Test oracle” problem

- Software Testing: A Research Travelogue (2000–2014), FOSE’14
 - “Overall, although there have been some **initial research** efforts in this direction, the problem of constructing oracles in an automated or semi-automated fashion is **still by and large open**.”
- Automated Test Oracles: A Survey
 - Mauro Pezzè; Cheng Zhang, Switzerland and Dipartimento di Informatica
- The Oracle Problem in Software Testing: A Survey
 - IEEE Transactions on Software Engineering 2015 vol.41 Issue No.05

Challenge: “Test oracle” problem, cont.

- test oracles can be specified
- test oracles can be derived
 - invariant, docs, regression
- test oracles can be built from implicit information
 - assert, failure
- no automatable oracle is available, yet it is still possible to reduce human effort

Challenge: “Test oracle” problem, cont.



Approaches without Symbolic Execution

- Split existing test cases
 - split every assert
- Test suite optimization
 - reduce test suite size

```
@Test
public void testIsEndedBy() {
    assertFalse(intRange.isEndedBy(null));

    assertFalse(intRange.isEndedBy(5));
    assertFalse(intRange.isEndedBy(10));
    assertFalse(intRange.isEndedBy(15));
    assertTrue(intRange.isEndedBy(20));
    assertFalse(intRange.isEndedBy(25));
}

@Test
public void testIsBefore() {
    assertFalse(intRange.isBefore(null));

    assertFalse(intRange.isBefore(5));
    assertFalse(intRange.isBefore(10));
    assertFalse(intRange.isBefore(15));
    assertFalse(intRange.isBefore(20));
    assertTrue(intRange.isBefore(25));
}
```

Thanks

Questions?